A Subdivision Approach to Planar Semi-algebraic Sets

Angelos Mantzaflaris and Bernard Mourrain

GALAAD, INRIA Méditerranée
BP 93, 06902 Sophia-Antipolis, France
[FirstName.LastName]@inria.fr
http://www-sop.inria.fr/galaad

Abstract. Semi-algebraic sets occur naturally when dealing with implicit models and boolean operations between them. In this work we present an algorithm to efficiently and in a *certified* way compute the connected components of semi-algebraic sets given by intersection or union of conjunctions of bi-variate equalities and inequalities. For any given precision, this algorithm can also provide a polygonal and isotopic approximation of the exact set. The idea is to localize the *boundary curves* by subdividing the space and then deduce their shape within small enough cells using only boundary information. Then a systematic traversal of the boundary curve graph yields polygonal regions *isotopic* to the connected components of the semi-algebraic set. Space subdivision is supported by a kd-tree structure and localization is done using Bernstein representation. We conclude by demonstrating our C++ implementation in the CAS Mathemagix.

Key words: subdivision algorithm, semi-algebraic set, connected component, algebraic curve, topology computation

1 Introduction

Planar semi-algebraic sets are unions of subsets $\mathcal{S}$ of $\mathbb{R}^2$ that satisfy a set of bivariate polynomial equalities and inequalities. These sets appear naturally when polynomial constraints are used for instance to describe regions of validity for a physical problem. Piecewise algebraic representation of shapes is commonly used in Computer Aided Geometric Design, for instance in B-spline parametric representation of curves, or even surfaces of volumes, that also belong to the class of real semi-algebraic sets. Constructive Solid Geometry models also used in CAGD are semi-algebraic sets if the involved solid primitives are algebraic. In domains such as optimization, an important problem is the computation of global optimum of (polynomial) functions under (polynomial) constraints. These constraints define a semi-algebraic set as the solution space, in which the optimal points will be searched [15], [12]. In other words, semi-algebraic sets provide a general framework to handle many shape representations that are commonly used in Shape Modeling.

In the present paper we present a new technique to handle semi-algebraic sets in the plane. We note that our method can be extended to dimension three, without theoretical obstacles. Indeed, the implementation is done in a generic programming framework that allows extension to dimension three without relatively little additional effort, since abstract types and templated data structures are heavily used.

The study of real semi-algebraic sets has a long historical background [18], with important theoretical contributions for instance on their triangulation [11], [13]. More algorithmic questions have also been tackled, essentially using the well-known Cylindrical Algebraic Decomposition [6]. This approach is based on performing successive projections of semi-algebraic sets onto subspaces of dimension one less and then lifting back to the projected set. It yields a decomposition of a semi-algebraic set S into (connected) components, defined by sign conditions deduced from some "subresultant" polynomial sequences [5], [8], [3].

One of the bottlenecks for practical applications of C.A.D.-based approaches, even in small dimension, is its double exponential complexity behavior. This is due mainly to computations with algebraic numbers of possibly high degree. Other obstacles include the lack of extension to approximate computation, required by applications in CAGD and the problem of robust description of the components. Our approach refrains from costly algebraic manipulations, hence avoids the high complexity of exact computation. It is based on real root isolation techniques, which are well suited for approximate yet certified computations. Moreover, it gives an answer to the problem of representing the semi-algebraic set in a way that is both topologically correct and suitable for applications. This overcomes the inflexible description by sign conditions or other implicit descriptions, for instance the one in [2], where each connected component is described itself as a semi-algebraic set.

We propose a subdivision approach that concentrates on rectangular domains of $\mathbb{R}^2$ and computes a piecewise linear approximation of a semi-algebraic set in the domain, which is topologically equivalent to it. The defining equations of the set are transformed to tensor-Bernstein form. This gives a numerically stable way to subdivide this representation into sub-domains, until certain regularity conditions are fulfilled. During the subdivision process the cells that touch the boundary of the semi-algebraic set are identified and their adjacency structure is represented as a graph. When this process terminates, we follow this graph to recover contours that define the geometry of the set. A tolerance $\varepsilon > 0$, given in the input, controls the precision of the computed approximation. Nevertheless, the regularity conditions imply a topologically correct result. In this sence, the algorithm extends the approach in [1] on the topology of algebraic curves, by providing a efficient way to deal with semi-algebraic regions and to perform boolean operations on these regions.

We start by defining the family of sets that we are interested in.

Definition 1. *The family* $\mathfrak{S} \subseteq 2^{\mathbb{R}^2}$ *of* semi-algebraic sets *is the closure under union and intersection of subsets of* $\mathbb{R}^2$ *of the form*

$$\left\{(x,y) \in \mathbb{R}^2 \,:\, f(x,y) = 0\right\} \quad \textit{and} \quad \left\{(x,y) \in \mathbb{R}^2 \,:\, g(x,y) > 0\right\}$$

where f, $g \in \mathbb{R}[x, y]$.

We call the above sets *basic semi-algebraic sets*. These definitions extend naturally to higher dimension.

If $\mathcal{S} \in \mathfrak{S}$, its complement $\mathcal{S}^c = \mathbb{R}^2 \backslash \mathcal{S}$ is easily seen to belong to $\mathfrak{S}$. The family $\mathfrak{S}$ is thus stable by intersection, union and complementary. Another important property of semi-algebraic sets is that the projection of a semi-algebraic set is a semi-algebraic set [3].

Our algorithm has as input an *initial frame* $\mathcal{D}_0 = [a, b] \times [c, d]$ and a semi-algebraic set $\mathcal{S}$, given in *disjunctive normal form*, that is, in the form $\mathcal{S}_1 \cup \cdots \cup \mathcal{S}_k$ where each $\mathcal{S}_i$ is an intersection of basic semi-algebraic sets, hence defined as a subset $\{(x, y) \in \mathbb{R}^2 : g_1 = 0, \ldots, g_m = 0, f_1 > 0, \ldots, f_n > 0\}$. It outputs a boundary effective representation of the connected components of this semi-algebraic set.

Given a precision $\varepsilon > 0$, it can also output a polygonal approximation of the set inside the domain $\mathcal{D}$, within the precision ε, which moreover is isotopic to S in the following sense:

Definition 2. *Two semi-algebraic sets* $\mathcal{S}_1, \mathcal{S}_2$ *of* $\mathbb{R}^2$ *are* isotopic *if there exists a continuous application* $F : \mathbb{R}^2 \times [0, 1] \mapsto \mathbb{R}^2$ *such that* $F|_{t=0}$ *is the identity map,* $F(\mathcal{S}_1, 1) = \mathcal{S}_2$ *and for all* $t \in [0, 1]$, $F|_t : \mathbb{R}^2 \mapsto \mathrm{Im}F|_t$ *is a homeomorphism.*

We introduce some notation. Throughout the text $\mathcal{S}$ will refer to an input semi-algebraic set. By a slight abuse of notation we might denote by $\mathcal{S}$ both the semi-algebraic set and the set of underlying defining polynomials. The meaning will be clear from the context. Let f be a polynomial of the input. We refer to parts of the real algebraic curve $f = 0$ that belong to $\partial\mathcal{S}$, the boundary of $\mathcal{S}$, as *boundary curves*. Points where boundary curves intersect (or a single boundary branch, part of some $f = 0$ is self-intersecting), are called *crossing points*. Also, we will refer to a *branch* of a curve, defined by two endpoints p, q, as the part of the curve between these points, e.g. the image of a continuous parametrized curve $r : [0, 1] \to \mathbb{R}^2$ s.t. $r(0) = p$, $r(1) = q$ and $f \circ r = 0$.

This paper is organized as follows: In Sect. 2 we provide details on the representation of the main objects in memory. Then in Sect. 3 we describe a subdivision process that computes a collection of cells covering $\partial\mathcal{S}$. This representation is used to compute the connected regions of $\mathcal{S}$, in Sect. 4. We specialize the main functions that appear in the algorithm first for the case of basic sets, in Sect. 5 and then for a general set of $\mathfrak{S}$ in Sect. 6. We conclude with examples and an overview of our implementation in Sect. 7.

2 Representation

We begin by describing the main objects in the algorithm, called hereafter cells, and how they are represented in memory.

A *cell* carries local information for $\mathcal{S}$ in a rectangular domain $\mathcal{D} = [a, b] \times [c, d]$. This information includes the Bernstein representation over $\mathcal{D}$ of the defining

equations of $\mathcal{S}_i$, whenever $\mathcal{S}_i \cap \mathcal{C} \neq \varnothing$. It also carries the intersections of every branch of $\partial\mathcal{S}$ that crosses the cell with the cell frame $\partial\mathcal{C}$. The cells of interest are exactly the cells that contain branches of boundary curves, i.e. parts of $\partial\mathcal{S}$. These cells are identified during the subdivision process.

A local description of $\mathcal{S}$ in a cell is achieved using the tensor-Bernstein representation over $\mathcal{D}$ of every polynomial that defines $\mathcal{S}$. This representation is computed using DeCasteljau's algorithm. It yields for $f \in \mathbb{R}[x, y]$, an expansion

$$f(x,y) = \sum_{i=0}^{d_x} \sum_{j=0}^{d_y} \gamma_{i,j}\, B^i_{d_x}(x;a,b)\, B^j_{d_y}(y;c,d) \ ,$$

where d_x, d_y is the degree of f in x, y resp. and $B^i_{d_x}(x;a,b)$ the $i-$th Bernstein polynomial of degree d_x over the interval $[a,b]$, namely $B^i_{d_x}(x;a,b) = \binom{d_x}{i}(x-a)^i(b-x)^{d_x-i}(b-a)^{-d_x}$, $0 \leq i \leq d_x$, $b < a$. Consequently we store an $(d_x+1)\times(d_y+1)$ matrix in memory to represent f, i.e. a dense Bernstein representation. A number of properties of this basis, e.g. convexity, variation diminishing, positivity etc, make it suitable for stable approximate computations. See [10] for more information.

The first cell $\mathcal{C}$ that is computed as soon as the algorithm is launched is the one corresponding to the initial frame $\mathcal{D}_0$. This initial cell carries all the polynomials of the input. When a sub-cell is computed, if $\partial\mathcal{S}_i$ does not cross that cell, for some i, $\mathcal{S} = \mathcal{S}_1 \cup \cdots \cup \mathcal{S}_r$, then the polynomials of $\mathcal{S}_i$ are not kept in the representation of it.

Another object is the *region*, which is a linear approximation of a 2-dimensional connected component of the semi-algebraic set. It is described as a collection of *contours*, that are closed loops properly oriented to delimit the region: The outer contour, or *shell*, is oriented counter-clockwise (CCW for short) whereas any internal contours, or *holes* are clockwise (CW) oriented. See Fig. 4 for a region defined by three contours.

Every contour is essentially a simple polygon described as a list of vertices that lie on the boundary of the exact set.

We also employ graph structures to keep adjacency information between cells. These are internally saved in memory using adjacency-list representation [7].

More specifically, we compute an undirected graph $\mathcal{A}$, in which the points where $\partial\mathcal{S}$ intersects $\partial\mathcal{C}$ correspond to edges and subdivision cells $\mathcal{C}$ correspond to vertices. We shall compute the restriction of the semi-algebraic set in a given initial domain, thus the border of this domain is from a computational point of view a limit for the regions to compute. For this reason, we also keep a directed graph containing the cells where boundary curves touch the initial frame and the four corner cells of $\mathcal{D}_0$. This forms a CCW loop and is used to complete any open contours that touch the boundary.

The space subdivision is tracked using a $kd-$tree, rooted at $\mathcal{D}_0$. The leaves of this tree is a partition of $\mathcal{D}_0$ into cells. The inner nodes represent the sequence of subdivisions that took place.

Example. In Fig. 3(left), we have a partition of the domain into 8 regular cells. The semi-algebraic set is the grayed area, described by a single contour. Here the graph $\mathcal{A}$ is the closed path of cells 2,7,6,8,3,2. The border graph is the directed closed path 2,1,7,6,4,3,2.

3 Subdivision process

The subdivision of the initial domain into regular cells is a main operation of the algorithm. It consists in splitting the initial domain into smaller cells until certain local properties are satisfied. These properties will allow in a later step the construction of a topologically correct approximation of the (boundary of the) set in each cell.

During this process we construct a graph $\mathcal{A}$ whose vertices are the cells that span $\partial\mathcal{S}$. Alg. 3.1 presents the general process. Here a cell is regarded as an abstract object that supports the following operations:

• **Regularity test**(IsRegular). A cell is considered regular if the topology of $\mathcal{S}$ inside the cell is known, i.e. it can be deduced using only discrete data stored in the cell, namely the points in $\partial\mathcal{C} \cap \partial\mathcal{S}$, or even the sign of some derivatives on them. Hence interesting cases are the cells that contain branches of boundary curves. Some characteristic examples of this are presented in Fig. 1. If there is

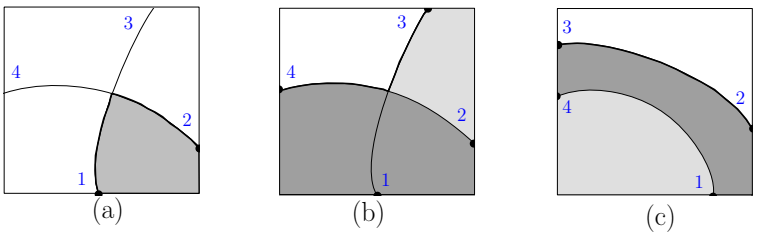

Fig. 1. Examples of cells that are regular and intersect $\mathcal{S}$: (a) intersection of two basic sets, (b) union of two basic sets with a crossing, (c) union of two sets.

more than one crossing point in the cell, that is, branches that intersect each other, then there is ambiguity on how the region behaves in the cell. Thus the regularity implies that we have at most one crossing point inside the cell and that the branches inside $\mathcal{C}$ have a monotone behavior. This behavior is connected to special points on the boundary curves, namely points with vertical or horizontal tangents.

• **Boundary curve intersection test**(OnBoundary). It is used to identify if a cell is intersecting $\partial\mathcal{S}$, i.e. $\mathcal{C} \cap \partial\mathcal{S} \neq \varnothing$. This can be done by inspecting the sign variations of Bernstein coefficients of the polynomials that define $\mathcal{S}$. *Descartes' Rule of Signs* implies that if there is a branch of $\partial\mathcal{S}$ in $\mathcal{C}$, then there will be sign

variations on the coefficients of some boundary equation. On the other hand, by the positivity property of the Bernstein basis, if the coefficients over a cell $\mathcal{C}$ of a curve $f = 0$ have no sign variations, then there cannot be a branch of this curve in $\mathcal{C}$.

Algorithm 3.1: Subdivision algorithm

Input: A cell $\mathcal{C}_0$ corresponding to the initial domain $\mathcal{D}_0$.
Output: A partition of $\mathcal{C}_0$ into regular cells and a cell graph $\mathcal{A}$.
Initiate a $kd-$tree $\mathcal{K}$ and set its root to $\mathcal{C}_0$;
Initiate a graph $\mathcal{A}$ with a vertex $\mathcal{C}_0$;
for *all unvisited leaves* $\mathcal{C}$ *in* $\mathcal{K}$ **do**
  if OnBoundary($\mathcal{C}$) *and not* IsRegular($\mathcal{C}$) **then**
    subdivide $\mathcal{C}$ into two children $\mathcal{C}_L$ and $\mathcal{C}_R$;
    put an edge in $\mathcal{A}$ between $\mathcal{C}_L$ and $\mathcal{C}_R$;
    distribute the $\mathcal{A}-$neighbors of $\mathcal{C}$ to $\mathcal{C}_L$, $\mathcal{C}_R$;
    remove $\mathcal{C}$ from $\mathcal{A}$;
  else
    mark $\mathcal{C}$ as visited;
  end
  return $\mathcal{K}$, $\mathcal{A}$;
end

During the subdivision process the following information is computed:

- Space partition information in the kd-tree structure.
- Local information in the subdivided cells: the tensor-Bernstein representation over the cell, critical points contained in the cell, intersection points of $\partial\mathcal{S}$ with the cell frame.
- Adjacency information between the cells, in horizontal and vertical direction. The cells in which the boundary curves touch the border $\partial\mathcal{D}_0$ are also connected in a counter-clockwise loop, to serve the purpose of limiting the computation inside $\mathcal{D}_0$.

• **Space Partition.** The cells that derive from successive subdivisions are organized in a kd-tree structure [4], rooted at the initial domain $\mathcal{D}_0$. The nodes in this tree have pointers to their left and right children, as well as to the parent node. The coordinate in which the subdivision takes place at every level of the tree is not fixed; it is implied every time by the dimensions of the current cell, thus at the same level of the tree we may have cell subdivisions either in x or y coordinate.

This structured partition allows to perform fast point location queries. The reason we have chosen a kd-tree rather than a quad-tree is economy wrt the overall number of cell subdivisions as well as the modularity that it offers, for instance it's direct adaptation to three or more dimensions.

There are two basic tests to be defined, to guide the subdivision process. The first identifies that a cell is *regular*, i.e. the topology of the semi-algebraic set in

the cell is known. In this case the subdivision stops at this branch of the kd-tree. The second test identifies if $\partial\mathcal{S}$ intersects the current cell. If not, then either $\mathcal{C} \subseteq \mathcal{S}$ or $\mathcal{C} \cap \mathcal{S} = \varnothing$, thus there is no need to subdivide it any further.

• **Cell subdivision.** Subdividing a cell $\mathcal{C}$ along some coordinate is essentially to compute, starting from the Bernstein representation over $\mathcal{C}$, representations over some sub-domains of $\mathcal{C}$. This operation is carried out by one call of DeCasteljau's algorithm [10].

Moreover, along the line where the splitting takes place, we solve a univariate Bernstein polynomial for every boundary curve that intersects the cell, in order to compute intersection with the new frame sides. The existing crossing points and frame intersection points are distributed to the resulting sub-cells, Fig. 2.

• **Adjacency graph update.** At each subdivision step, a former leaf of the kd-tree obtains two children. To update the cell graph, we disconnect this node and distribute it's neighbors to the new children, according to the direction of splitting. Finally, we introduce a new edge that joins the two children along the corresponding direction. These steps, demonstrated in Fig. 2, assure that at any point of the subdivision, the leaves of the kd-tree, which form a partition of $\mathcal{D}_0$, are connected to the neighboring cells in all four sides.

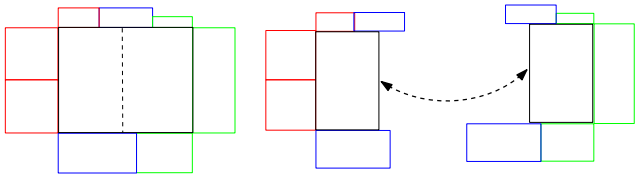

Fig. 2. Cell subdivision along $x-$direction. Neighbors of the parent (left) are distributed to the children(right). An edge is added between the latter.

4 Region recovery

In this section we explain how we pass from the cell description to a polygonal approximation of the (connected components of the) semi-algebraic set. We will demonstrate that as soon as the subdivision Alg. 3.1 terminates, we are able to recover the shape of the semi-algebraic set, and guarantee the correctness of the construction.

The output is a list of regions that correspond to connected components of the semi-algebraic set. The set of cells that intersect a region can readily provide a triangulation of the region, which can be outputted for use in rendering. Each region is represented as a set of closed oriented contours. The orientation of every contour reveals whether it is the exterior boundary, or *shell* of the region, or an internal gap, or a *hole*. There is a unique shell for every region of $\mathcal{S}$.

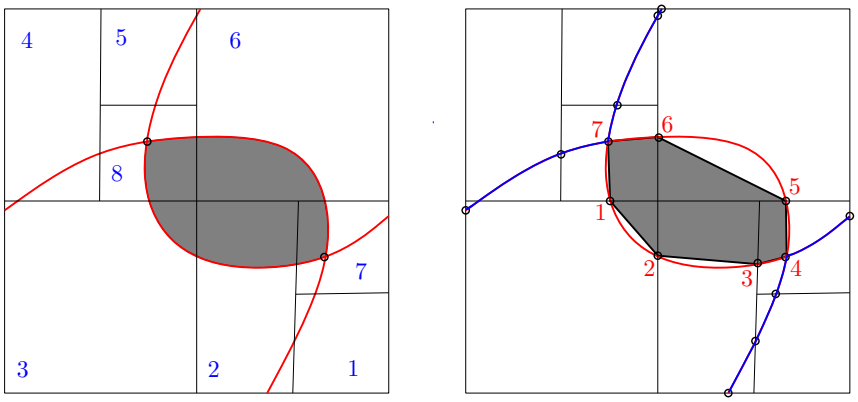

Fig. 3. Left: Subdivision process, with marked subdivided cells and intersections. Right: Computed polygonal region, marked with the oriented list of contour points.

To compute the regions, it suffices to traverse the cell graph $\mathcal{A}$ in a suitable way and recover the shell and holes of every region in the set. The algorithm for region computation is summarized in Alg. 4.1.

Algorithm 4.1: Region computation

Input: A cell graph $\mathcal{A}$ covering the semi-algebraic set $\mathcal{S}$.
Output: A list L of polygonal regions, one for every connected component of $\mathcal{S}$.
$L \leftarrow \varnothing$;
for *all boundary cells* $\mathcal{C}$ *in* $\mathcal{A}$ **do**
  if $\mathcal{C}$ *is not visited* **then**
    $F \leftarrow$ DiscoverContour($\mathcal{C}$);
    if F IsCCW **then**
      Initialize region R with F;
      push R to L;
    else
      attach hole F to it's containing shell
    end
  end
end
return L;

The orientation check IsCCW depends only on the contour F. Every closed contour can be assigned an orientation; if one walks around the curve in such a way as to keep the bounded region on one's left at all times, the contour is said to be positively oriented. If the contour is traversed in the opposite direction, then it is said to be negatively oriented. Let $c = (p_1, p_2, \ldots, p_n)$ with $p_i = (x_i, y_i)$, $p_{n+1} = p_1$ be a list of points defining a closed polygonal contour. The sign of the

quantity $\sum_{i=1}^{s}(x_i y_{i+1} - x_{i+1} y_i)$ determines weather c is positively or negatively oriented. This sum is twice the (signed) area of the contour.

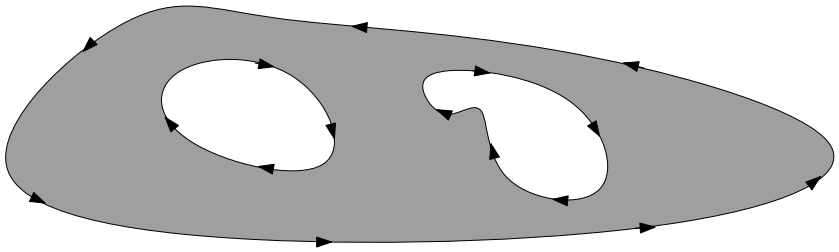

Fig. 4. A region defined by it's oriented border. All the contours are CCW-oriented wrt the grayed region. This leaves the holes CW oriented with respect to the bounded domain they define.

The function DiscoverContour, presented in Alg. 4.2, returns a contour that crosses the cell $\mathcal{C}$ and is oriented CCW wrt the region it delimits. For instance, both the holes and the shell of the region in Fig. 4 are CCW oriented wrt the grayed region. It is required that the cell argument is *regular*, so that the global shape of the contour can be determined by following the known local topology in the cell. This is ensured by the subdivision process of Sect. 3.

Apart from the cells containing branches of the boundary contours, there are special cells that are needed in order to constrain the computation in the initial frame $\mathcal{D}$. These are the boundary cells that touch the frame as well as the four corners of $\mathcal{D}$. They are connected in a CCW loop during the subdivision process that is used to complete the contours that escape $\mathcal{D}$ and would not be closed otherwise.

4.1 Following the boundary curves around a region

The main function in Alg. 4.1 is DiscoverContour, which is in turn based on two routines, Pair and StartingPoint.

• **Pair.** If a cell intersects both a region and the region's boundary, then for every intersection point p there is a unique point q that is connected to p via a segment of $\partial\mathcal{S}$ that lies inside the cell. If the cell in question is also regular, q can be computed using sign conditions along $\partial\mathcal{C}$. We define this point q to be the result of Pair$(\mathcal{C}, p)$. If this point q is different from p, then evidently it is connected to p via a branch of some boundary contour of the region. Alg. 4.3 presents a general strategy to compute q.

Example. In Fig. 1 the result of Pair is: (a)$1 \to 2$, (b)$4 \to 3$, (c)$2 \to 3$. Note that in case (c), the branch $1 \to 4$ will not occur in the computation, since it does not belong to $\partial\mathcal{S}$.

• **Starting Point.** A contour has to be traversed with the correct orientation, otherwise we would not be able to distinguish between shells and holes of a region. For this, it suffices to provide the first two points in the point list of the contour with the correct orientation. This is the task of the StartingPoint($\mathcal{C}$) routine. It returns a point p on $\partial\mathcal{C}$ s.t. the oriented branch with endpoints p, Pair(p) has on it's left side the region to be computed. This is a special case of the Pair computation described in the next paragraph.
Example. For Fig. 1 the result of StartingPoint is: (a)2, (b)3, (c)2. Indeed, the respective branches (a)2 $\rightarrow$ 1, (b)3 $\rightarrow$ 4 and (c)2 $\rightarrow$ 3, are CCW-oriented wrt $\mathcal{S}$.

Looking at the graph $\mathcal{A}$ induced by Alg. 3.1, we distinguish two kinds of regular cells:

- Cells that contain non-crossing branches of $\partial\mathcal{S}$.
- Cells that contain branches that intersect at one crossing point.

Recall that the outcome of Pair($\mathcal{C}$) is the point connected to p via a branch which lies inside $\mathcal{C}$. The general algorithm is presented in Alg. 4.3. The essential tool for this computation is an efficient way to check if a given point on $\partial\mathcal{C}$ is contained in $\mathcal{S}$. This is done using the sign of the Bernstein coefficients. For every polynomial f of $\mathcal{C}$, there are four *extreme* coefficients that are equal to it's value on the four corners of $\partial\mathcal{C}$. Now taking into account that the sign of f along $\partial\mathcal{C}$ alternates every time we pass a boundary intersection point, we can determine the sign on any point of $\partial\mathcal{C}$ by starting from an extreme coefficient and counting points along $\partial\mathcal{C}$, up to the desired point.

If there is one crossing point in $\mathcal{C}$ the topology of $\partial\mathcal{S}\cap\mathcal{C}$ is conic (Fig. 1(a,b)). To choose the correct pair of a given point on $\partial\mathcal{C}\cap\partial\mathcal{S}$, we check whether a $\partial\mathcal{C}-$neighborhood on the left of p belongs to $\mathcal{S}$ or on the right of p. We output accordingly the point on the side where the test was positive.

If there is no crossing point, (Fig. 1(c)) it suffices to return the other end of the branch that starts from p. We shall see in the sequel how this information is recovered on regular cells.
Example. In the case of Fig. 3(left) we execute Alg. 4.2. Starting from cell 2, we obtain a first point of the contour, using StartingPoint routine. Succesive calls of the pair function give the sequence of points shown in Fig. 3(right). The process stops when we reach the cell 2 again, thus completing the contour.

It remains to specialize these functions. We continue by doing so, first in the case of basic algebraic sets and then in the case of intersection and union.

5 The case of basic semi-algebraic sets

A basic semi-algebraic set is defined by one polynomial, $\mathcal{S} = \{(x, y) : f > 0\}$, or $\mathcal{S} = \{(x, y) : f = 0\}$. In both cases the treatment is quite the same, and depends on the boundary curve $f = 0$, hence we shall suppose $\mathcal{S} = \{(x, y) : f > 0\}$. In the case of equality it is only the contour lines that will be outputted rather

Algorithm 4.2: DiscoverContour($\mathcal{C}$)

Input: A *regular* cell $\mathcal{C}$ of $\mathcal{A}$.
Output: A list F of points in the plane that define a closed contour.
$p \leftarrow$ StartingPoint($\mathcal{C}$);
Initialize a contour F and push p to it;
$\mathcal{C}_0 \leftarrow \mathcal{C}$;
repeat
  mark $\mathcal{C}$ as visited;
  $p \leftarrow$ Pair($\mathcal{C}, p$);
  push p to contour F;
  $\mathcal{C} \leftarrow$ the $\mathcal{A}-$neighbor of $\mathcal{C}$ that contains p;
until $\mathcal{C} = \mathcal{C}_0$;
return F;

Algorithm 4.3: Pair

Input: A regular cell $\mathcal{C}$ and an intersection point p on $\partial\mathcal{C}$.
Output: The intersection point q such that $\{p, q\}$ lie on a branch of $\partial\mathcal{S}$.
if *there is a crossing in* $\mathcal{C}$ **then**
  Let l, r be the CCW previous and next point, resp., of p, in $\partial\mathcal{C} \cap \{f = 0\}$;
  Based on which of the segments $\overline{lp}$ or $\overline{pr}$ lies in $\mathcal{S}$, return either l or r ;
else
  return the other end of the $\mathcal{C}-$branch starting from p;
end

than two-dimensional regions. After fully treating this case, we shall generalize by extending the operations to the cases of intersection and union.

This case is closely related to the topology computation of an implicit real algebraic curve. The latter is the partition of space into points, edges and faces defined by the curve $f = 0$. See Figure 5 for an example. Note that recovering the topology of the real algebraic curve $f = 0$ is a special case of our algorithm. Indeed, it suffices to execute the subdivision algorithm on $\mathcal{S} = \{f = 0\}$ and then run the region recovery twice, once with $\mathcal{S} = \{f > 0\}$ and once with $\mathcal{S} = \{-f > 0\}$. The union of these two outputs is exactly the set of faces defined by the curve $f = 0$.

5.1 Regularity test

We describe the regularity criteria that are used for the boundary curve of the set. We shall provide a brief overview and refer the reader to [1, Sect. 4] for an extended presentation.

The regularity depends on special points on the curve, that reveal the local shape of the curve in a neighborhood around them. These are:

Definition 3. *The set of* extremal points *of* $f \in \mathbb{R}[x, y]$ *is the solutions of the system* $\partial_x f(x, y) = \partial_y f(x, y) = 0$.

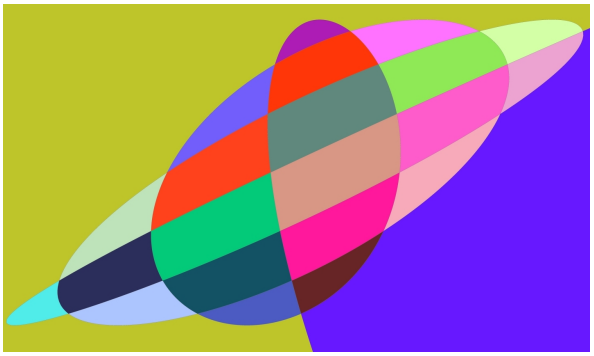

Fig. 5. The 23 faces in the topology of the degree 8 curve $f = 2 + 7x - 7y - 14x^3 + 7x^5 - x^7 - 16y^2 + 14y^3 + 20y^4 - 7y^5 - 8y^6 + y^7 + y^8 - 42y^2x - 70y^3x^2 + 35xy^4 + 70y^2x^3 + 42yx^2 - 35x^3y^4 + 7x^6y - 21x^5y^2 - 35x^4y + 21x^2y^5 + 35y^3x^4 - 7xy^6$ computed by running our algorithm on $\mathcal{S} = \{f > 0\}$, $\mathcal{S} = \{f < 0\}$ and $\mathcal{D} = [-4, 4] \times [-3, 3]$.

The set of singular points *of f is the subset of extremal points that also satisfy the equation $f(x, y) = 0$.*
The set of x-critical *(*y-critical*) points of f is the solution set of $\partial_x f(x, y) = f(x, y) = 0\}$ $(\partial_y f(x, y) = f(x, y) = 0\})$.*

Computing these points, approximately but also efficiently, is a vital ingredient of the algorithm. In [16], an algorithm is presented that acts on polynomials in Bernstein form. It uses domain subdivision as well as enveloping and preconditioning techniques to provide a robust polynomial solver. We rely on this solver to obtain good approximations of the points in Def. 3. These points are precomputed and during the subdivision process they are isolated between the cells, i.e. we do not allow more than one of them in a single cell. As a result, after the subdivision process terminates, we obtain a partition of $\mathcal{D}_0$ into regular cells of the following type:

- *x-regular* cells, those that contain no x-critical points (similarly for $y-$regular).
- *simply singular* cells, that contain a single singular point and all branches of $\partial\mathcal{S} \cap \mathcal{C}$ intersect it.

• **Regular cells.** If a cell is $x-$regular, it contains a number of $x-$monotone branches. In short, the direction of the tangential gradient vector $(\partial_y f, -\partial_x f)$ evaluated at the points in $\partial\mathcal{C} \cap \partial\mathcal{S}$ yields the connection of the branches inside $\mathcal{C}$. The Bernstein representation of the derivatives themselves are easily computed, since they are given as differences of Bernstein coefficients of f. A sufficient condition for f to be x-regular is that the Bernstein coefficients of $\partial_x f$ maintains a constant sign. By Descartes' law, this statement implies that the sign variations in $x-$direction should be at most one.

Note that in special cases where the critical point is on $\partial\mathcal{C}$ two branches may share a starting or ending point.

• **Simply singular cells.** If there is a single singular point in a cell $\mathcal{C}$, and no additional extremal points, one must test whether all the branches inside $\mathcal{C}$ cross this point. This would imply that the topology inside $\mathcal{C}$ is a cone starting from the singular point. The test is based on computing the topological degree, or Gauss map [17] of the vector field $\nabla f = (\partial_y f, \partial_x f)$ around the closed curve $\partial\mathcal{C}$. This breaks down to isolating the real roots of $\partial_x f$ and $\partial_y f$ along $\partial\mathcal{C}$. Khimshiashvili's theorem [14] relates the number of branches that reach the singular point to the topological degree $\deg(\nabla f, \mathcal{C})$; it states that the number of branches is exactly $2\,(1 - \deg(\nabla f, \mathcal{C}))$. If this number coincides with the cardinality of $\partial\mathcal{C} \cap \partial\mathcal{S}$ then we can treat this cell, otherwise there are additional branches in $\mathcal{C}$ and the subdivision will continue until they are isolated from the singular point.

6 The general case

To treat semi-algebraic sets with more than one defining equation, it suffices to extend the main operations in this case. Our aim is to have a covering of the boundary curves of $\partial\mathcal{S}$ by regular cells. The main difference is that crossing branches in a cell can correspond to two basic sets in a union, or two basic sets in an intersection. Treating correctly these cases will extend our algorithm to the whole family of semi-algebraic sets. Again, we assume that the basic sets are defined by inequalities, since restricting to (in the case of intersection) or attaching (in the case of union) a curve segment to the output is not essentially different from treating boundary curves of two dimensional components. In particular, the cell graph $\mathcal{A}$ that we obtain from the subdivision Alg. 3.1 will span any components of lower dimensions.

Let $\mathcal{S} = \mathcal{S}_1 \cup \cdots \cup \mathcal{S}_k$. Recall that a cell $\mathcal{C}$ carries the polynomials of $\mathcal{S}_i$ if $\partial\mathcal{S}_i \cap \partial\mathcal{C} \neq \varnothing$. For all the other parts $\mathcal{S}_j$, it is either $\mathcal{S}_j \cap \mathcal{C} = \varnothing$ or $\mathcal{C} \subseteq \mathcal{S}_j$, hence $\mathcal{C}$ does not interfere with the boundary curves of these components.

We define a regular cell to be a cell in which every attached polynomial is regular (in the sense of Sect. 5.1) and conforms to any of the following properties:

1. There is only one set $\mathcal{S}_i$ in $\mathcal{C}$ and at most one (self-)intersection.
2. There are two sets $\mathcal{S}_i$ and $\mathcal{S}_j$ and one intersection between a branch of $f \in \mathcal{S}_i$ and $g \in \mathcal{S}_j$.

These intersection points are also computed using the Bernstein solver [16] and are isolated among the cells during the subdivision process.

Deciding if a region spans $\partial\mathcal{S}$ is done by checking whether it belongs to the boundary of every $\mathcal{S}_i$ that is carried by $\mathcal{C}$, and consists again in checking signs on the boundary.

To simplify the process, we rely on basic cells (cells that have branches of a single basic set contributing to $\mathcal{S}$) for determining the orientation of regions, i.e. applying StartingPoint. This is a mild assumption, since in any case, boundary curves away from crossings define basic semi-algebraic sets. This assumption also simplifies the way we deal with cells like Fig. 1(c), since we only need to

know the connection inside the cell in order to traverse them and choose the correct branch (for instance, in Fig. 1(c), discard the locally redundant curve).

We describe how we compute PAIR in the above two cases:

• **Case 1.** There is a set of branches in the cell that intersect in one point only, similar to 1(b). Since the corresponding basic sets are combined by intersection we search around $\partial\mathcal{C}$ for a part that attains positive sign on all involved polynomials, to decide the PAIR routine.

• **Case 2.** Two branches intersect, corresponding to basic sets combined by union, for instance 1(c). We propagate the search to points around parts of $\partial\mathcal{C}$ that satisfy any of the sign conditions implied by $\mathcal{S}_i$ or $\mathcal{S}_j$. When we reach a part that is outside $\mathcal{S}$, we return the last point found.

7 Implementation and demonstration

Our implementation is generic, working on abstract classes of cells, that define internally a small number of predicates. We chose to use the open-source project MATHEMAGIX[1], for the fast data structures it provides for polynomials and it's support to certified arithmetic primitives. Our code is written in the frame of the `shape` module, which is the part of MATHEMAGIX providing a variety of geometric operations in two or three dimensions.

Solution of univariate and bi-variate systems of polynomial is performed using the algorithm in [16], which is hosted in the module `realroot`. This module also provides algebraic operations, Bernstein dense representation and a variety of zero-dimensional system solvers. Hardware accelerated rendering of output has been made possible using AXEL[2] platform.

A first example is given in Fig. 6, where we can see the cells deduced by the subdivision process together with the defining curves (left), and the computed regions (right) based on this cell graph. The boxes span only the actual boundary curves of $\partial\mathcal{S}$, but we also draw the full defining curves to give an idea of the situation.

A precision of $\varepsilon = 0.05$ is used, that is, the cells are subdivided down to this size, to obtain a smooth visual result. Note the two branches that are almost tangent near the bottom left corner. They cause the subdivision to continue further around this area until the branches are properly separated.

In Fig. 7 we compute a set $\mathcal{S} = \{(x, y) : f_1 > 0, f_2 > 0\}$ defined by a degree 6 and a degree 32 polynomial. The domain of computation is $[-1.5, 1.5]^2$ and precision set as before, $\varepsilon = 0.05$. The running time for this example is less than one second. Our implementation is able to handle polynomials of quite higher degree, up to 100 or more. Here the resulting regions contain holes, which are correctly recognized. Finally, Fig. 7 presents the complementary set, $\mathcal{S}^c = \{(x, y) : -f_1 > 0\} \cup \{(x, y) : -f_2 > 0\}$ given by 4 connected components.

The purpose of the third example is to demonstrate how our implementation can handle degenerate cases, namely cusps. We treat a single curve of degree 28,

[1] http://www.mathemagix.org
[2] http://axel.inria.fr

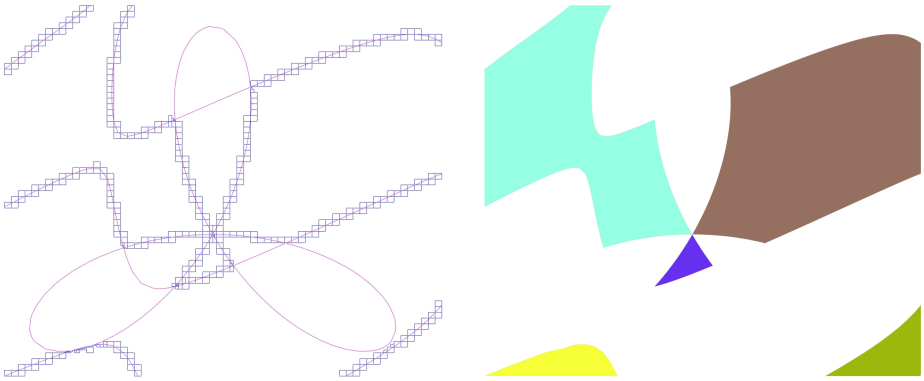

Fig. 6. $\mathcal{S} = \{(x, y) : f_1 > 0, f_2 > 0\}$ with $f_1 = x^4 + 2x^2y^2 + y^4 + 3x^2y - y^3$, $f_2 = -105y^2x^4 - 80y^3 + 140x^3y^3 - 140y^3x + 35y^4 - 105y^4x^2 + 48y^5 + 42xy^5 - 42x^2 + 35x^4 - 7x^6 + 32y + 84xy - 140x^3y + 42x^5y + 210x^2y^2 - 42y^2 - 7y^6 - 8y^7 + 7$ over the box $[-1, 1]^2$.

having several cusps. This curve is taken from a real application in non-linear computational geometry, namely the computation of the Voronoi diagram of ellipses, see recent paper [9]. We compute all regions defined by the curve, in the domain $[-7, 3]^2$ and set precision to $\varepsilon = 0.5$. Detailed output is shown in Fig. 9.

Acknowledgments. The research leading to these results has received funding from the European Community's Seventh Framework Programme [FP7/2007-2013], Marie Curie ITN SAGA, grant agreement n° [PITN-GA-2008-214584].

References

1. Alberti, L., Mourrain, B., Wintz, J.: Topology and Arrangement Computation of Semi-algebraic Planar Curves. *Comput. Aided Geom. Des.*, 25(8), 631–651, (2008)
2. S. Basu, R. Pollack, and M.-F. Roy. Complexity of computing semi-algebraic descriptions of the connected components of a semi-algebraic set. In: ISSAC '98: Proceedings of the 1998 international symposium on Symbolic and algebraic computation, pp. 25–29, ACM, New York, USA (1998)
3. Basu, S., Pollack R., Roy, M.-F.: *Algorithms in Real Algebraic Geometry*. Springer-Verlag, Berlin (2003)
4. Bentley, J. L.: Multidimensional divide-and-conquer. *Commun. ACM*, 23(4), pp. 214–229 (1980)
5. J. Bochnak, M. Coste, and M.-F. Roy.: *Géométrie Algébrique Réelle*. Springer-Verlag, Heidelberg (1987)
6. Collins, G. E.: Quantifier elimination for real closed fields by cylindrical algebraic decomposition. In: 2nd GI Conf. on Automata Theory and Formal Languages, LNCS, vol. 33, pp. 134–183, Springer-Verlag (1975)
7. Cormen, T. H., Leiserson, C. E., Rivest, R. L., Stein. C.: Introduction to Algorithms. MIT Press, Cambridge, second edition (2001)

Fig. 7. Left: defining curves and cell graph. Right: boundary contours of the underlying set.

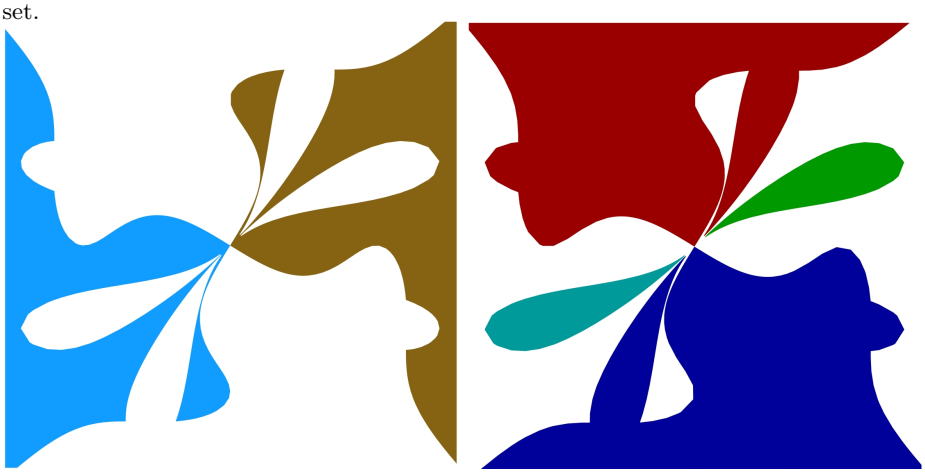

Fig. 8. Left: Two connected components of a semi-algebraic set, each containing a hole. Right: regions of the complementary set.

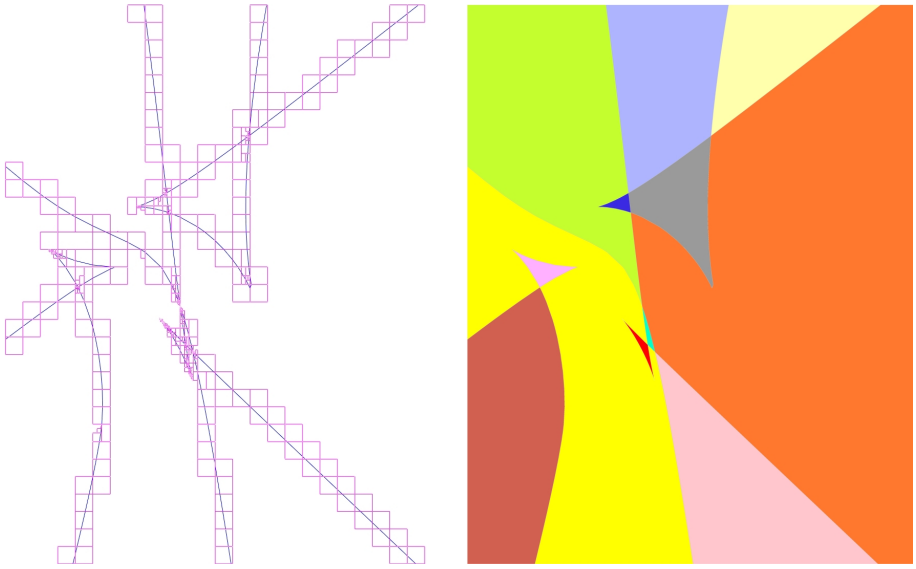

Fig. 9. Computing the topology of a degree 28 algebraic curve with cusps.

8. Coste, M.: An introduction to semi-algebraic geometry. RAAG network school (2002)
9. Emiris,I. Z., Tsigaridas, E. P., Tzoumas, G. M.: Exact delaunay graph of smooth convex pseudo-circles: general predicates, and implementation for ellipses. In: SPM '09: 2009 SIAM/ACM Joint Conference on Geometric and Physical Modeling, ACM, pp. 211–222, New York, USA (2009)
10. Farin, G.: Curves and Surfaces for CAGD: A Practical Guide. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (2002)
11. Hardt, R. M.: Triangulation of Subanalytic Sets and proper light subanalytic maps. Invent. Math., 38(3), 207–217 (1976/77)
12. Henrion, D. , Lasserre, J.-B., Lofberg, J.: GloptiPoly 3: Moments, Optimization and Semidefinite Programming. Optimization Methods and Software, 24(4-5), pp. 761–779 (2009)
13. Hironaka, H.: Triangulations of algebraic sets. In: Algebraic geometry (Proc. Sympos. Pure Math., Vol. 29, Humboldt State Univ., Arcata, Calif., 1974), pp. 165–185, Amer. Math. Soc., Providence, R.I. (1975)
14. Khimšiašvili, G. N.: The local degree of a smooth mapping. Sakharth. SSR Mecn. Akad. Moambe, 85(2), pp. 309–312 (1977)
15. Lasserre, J. B.: Moments, Positive Polynomials and their Applications, Vol. 1 of Optimization Series, Imperial College Press (2009)
16. Mourrain, B., Pavone, J. P.: Subdivision methods for solving polynomial equations. J. Symb. Comput., 44(3), pp. 292–306 (2009)
17. Stenger, F.: Computing the topological degree of a mapping in $\mathbb{R}^n$. Numer. Math., 25(1), pp. 23–38 (1975)
18. Tarski., A.: A decision method for elementary algebra and geometry. Univ. of California Press, Berkeley (1951)

Fig. 10. Semi-algebraic set defined by: $f_1 = -105y^2x^4 - 80y^3 + 140x^3y^3 - 140y^3x + 35y^4 - 105y^4x^2 + 48y^5 + 42xy^5 - 42x^2 + 35x^4 - 7x^6 + 32y + 84xy - 140x^3y + 42x^5y + 210x^2y^2 - 42y^2 - 7y^6 - 8y^7 + 7$, $f_2 = x^2 + 3y^2 - 1$, $f_3 = x^6 + y^2x^4 - y^4x^2 - 2x^4 - y^6 + 2y^4 + x^2 - y^2 + xy$ in domain $[-3, 3]^2$.

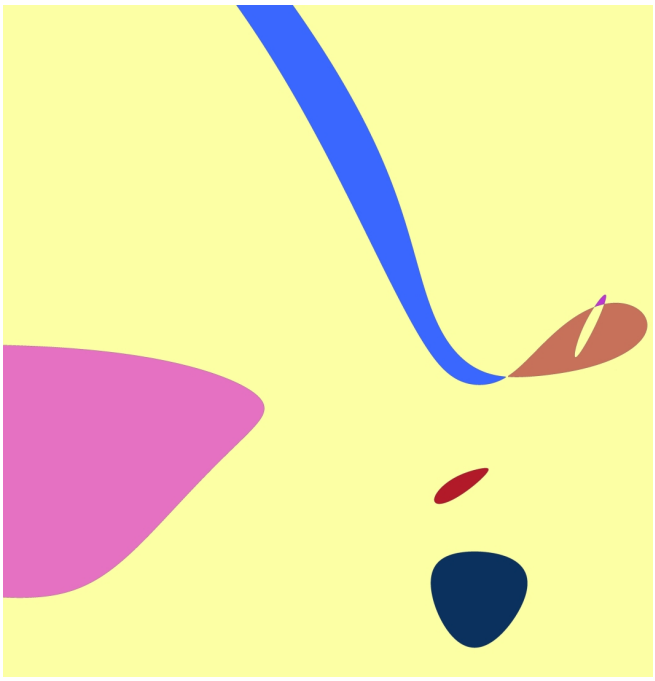

Fig. 11. Topology of a degree 76 curve coming from the self-intersection locus of a 3D surface.

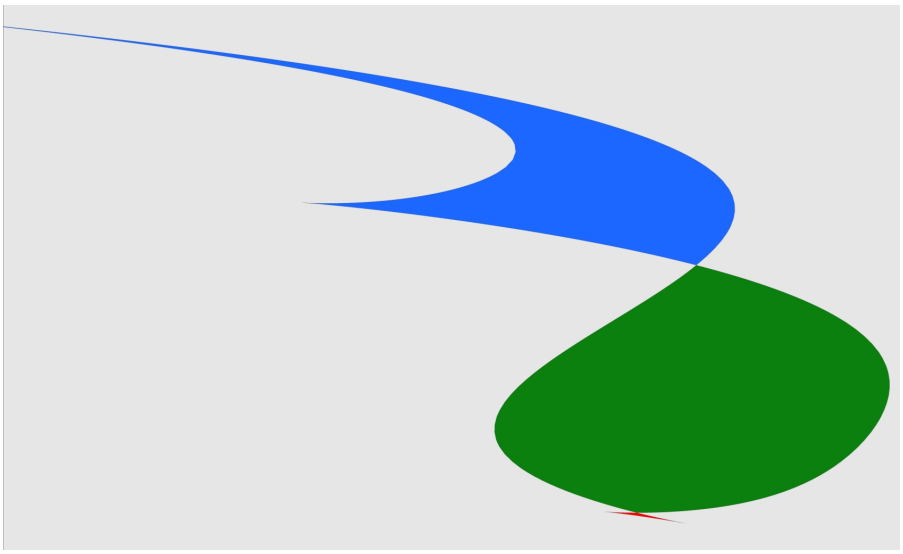

Fig. 12. A (degree 12) apparent contour of 3D surface with cusps.

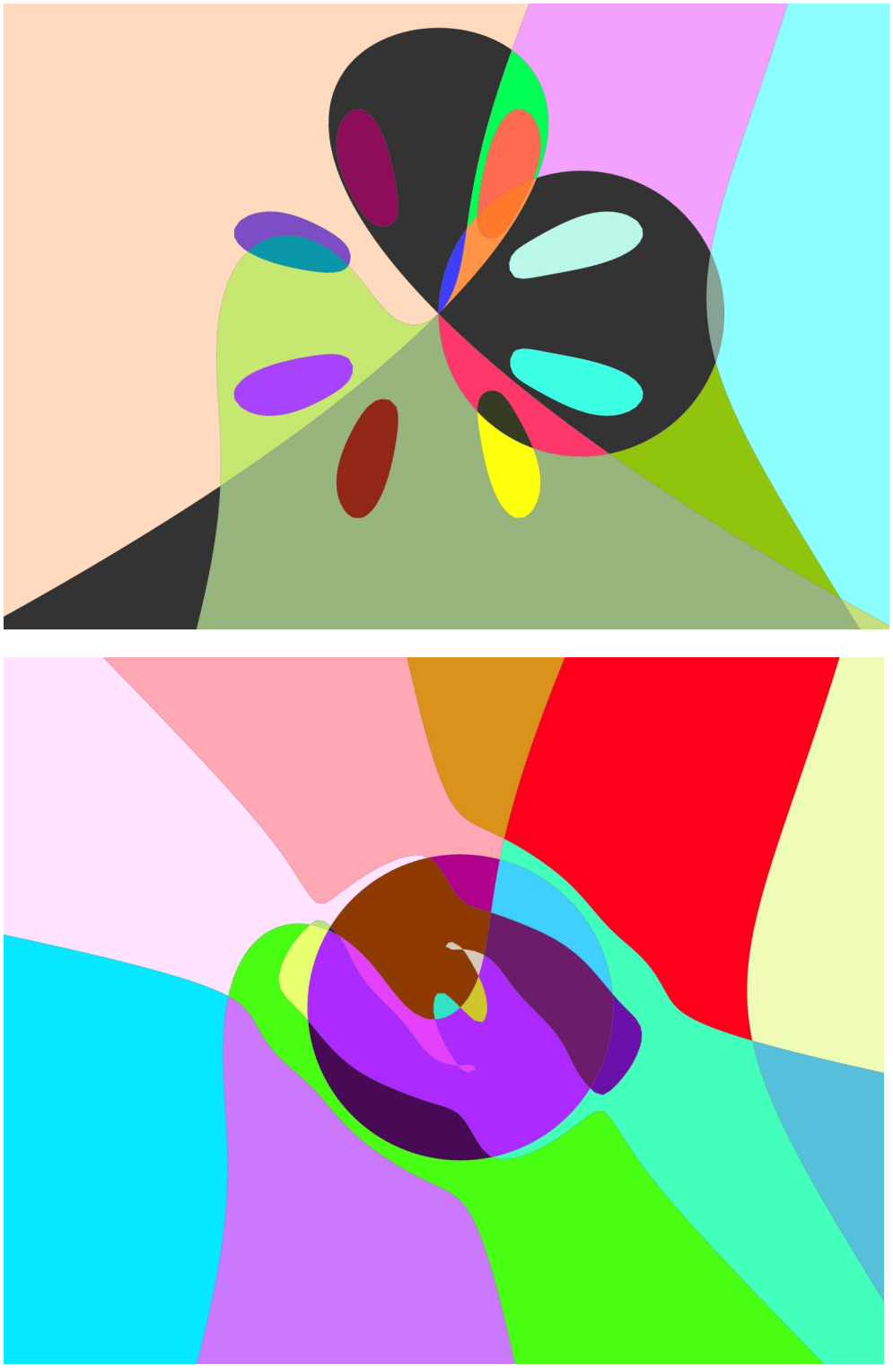

Fig. 13. Regions in the arrangement of three curves, of resp. degrees 32,4,4(top), 32,4,13 (bottom) computed using our algorithm on the underlying semi-algebraic domains.